

MEMORIA PRÁCTICA 1

ALGORITMO A ESTRELLA



U N I V E R S I D A D
COMPLUTENSE
M A D R I D

Marzo 2022

ALVARO LLORET BODELON
SERGIO TAPIA ORTEGA



ÍNDICE

Descripción.....	3
Ampliaciones.....	6
Manual de usuario.....	7
Ejemplos de uso.....	9

DESCRIPCIÓN

El lenguaje que hemos utilizado para la implementación del algoritmo es JavaScript, y HTML y CSS para la representación del tablero y sus elementos (inicio, fin, obstáculos, camino solución, etc.). Además para la comunicación entre la vista (el documento HTML) y la implementación (los documentos JavaScript) hemos usado una librería de JQuery.

La implementación está dividida en 5 archivos:

- Index.html, que como ya hemos dicho anteriormente contiene la representación del tablero junto con sus elementos. Está dividido en dos partes principales: un elemento <div> que contiene una tabla donde estará el tablero y sus elementos. Y otro <div> que encapsula un panel en cual se muestran las distintas acciones que puede realizar el usuario, tales como, seleccionar el tipo de casilla que vas a marcar (inicio, fin, obstáculos) o los botones de resetear y calcular.
- AEstrella.js contiene la implementación del algoritmo A estrella, con sus variables, objetos y funciones. Más adelante explicaremos a fondo el funcionamiento de este fichero.
- App.js, es el fichero que está conectado a la vista (index.html), y que tiene una serie de funciones diferentes para los distintos eventos que puede generar el usuario (calcular el recorrido, marcar una casilla, cambiar el modo, etc.). Además es el encargado de inicializar el tablero de dimensiones 10x10.
- Cordenada.js, en este archivo esta la clase coordenada, que es la que usamos para guardar la información de cada celda del tablero (fila, columna, la coordenada anterior, la fila del final, la columna del final, la distancia al nodo meta que la calculamos dentro de la misma clase, y la distancia al nodo actual la cual se le pasa por parámetro.
- PriorityQueue.js, por último, tenemos el archivo que contiene la clase PriorityQueue, que es una cola de prioridad que utilizamos para gestionar la lista que contiene los nodos abiertos del algoritmo. Funciona como cualquier cola prioridad de una librería externa.

Implementación del algoritmo A estrella (sin ampliaciones, más adelante explicaremos las ampliaciones añadidas) en el fichero AEstrella.js, esta clase consta de 8 variables: las dimensiones del tamaño del tablero (filas y columnas), una matriz de dimensiones del tablero, una cola de prioridad para almacenar los nodos abiertos, un array para almacenar los nodos cerrados, una coordenada de inicio que se pasa por parámetro, una coordenada de fin que se pasa por parámetro y un array de coordenadas que contiene la lista de obstáculos del tablero, también pasado por parámetro.

Además consta de 5 funciones:

- `iniciar()` es la primera que se llama desde `app.js`, y es la encargada de inicializar la matriz, rellena cada casilla con un string (“vacío” si es una casilla libre, “ini” si es la casilla de inicio, “fin” si es la casilla de fin, y “obstáculo” si en esa casilla corresponde un obstáculo).
- `Algoritmo()` es la función donde reside el algoritmo, empieza declarando un booleano “terminado” inicializado a falso y que indicará si se ha llegado a la casilla final, seguido de la creación de la coordenada inicial que posteriormente será introducida en la cola de prioridad “abierta”. A continuación comienza el bucle que terminará cuando el booleano terminado sea verdadero o la cola de nodos abierta este vacía. Sacamos de la cola de prioridad de nodos abiertos el elemento con mayor prioridad, lo guardamos en el array de nodos cerrados, y posteriormente recorreremos sus casillas adyacentes con dos bucles for. Primero comprobamos que no se salgan del tablero, y después miramos a través de la matriz con las coordenadas de las casillas adyacentes que no sean ni un obstáculo ni la casilla de inicio ni una casilla que ya esté abierta, seguido comprobamos que la casilla adyacente no sea el nodo padre de la casilla actual (con una función de esta misma clase), que la casilla adyacente no esté en el array de cerradas (con una función de esta misma clase). Si no cumple los requisitos anteriores seguimos en el bucle y pasamos a recorrer otra casilla adyacente. Si por el contrario sí que los cumple procedemos a calcular la distancia euclídea de la casilla adyacente a la actual. Una vez calculada hacemos la comprobación final a través de la matriz, si en las coordenadas de la casilla adyacente son el fin, creamos una coordenada con sus respectivas coordenadas y sumándole a la distancia del nodo actual la distancia que previamente habíamos calculado, asignamos el nodo actual como el anterior del adyacente y igualamos el booleano “terminado” a verdadero. Si no es la casilla final creamos otra coordenada de la misma manera que antes, asignándole también la coordenada actual como la anterior a la adyacente, la introducimos en la cola de prioridad “abierta” y la posición de la matriz correspondiente a las coordenadas de la casilla adyacente la rellenamos con “abierta”. Continuamos de la misma manera con las el

resto de coordenadas adyacentes hasta que no queden y entonces pasamos al nodo de más prioridad de la cola hasta encontrar el final o que no queden nodos en la cola de prioridad “abierta”.

- `Comparator(a, b)`, es el comparador que usamos para ordenar la cola de prioridad de nodos abiertos, el comparador devuelve de entre dos coordenadas la cual su suma de la distancia al nodo meta más la distancia al nodo actual sea menor.
- `ArraySolucion()`, es la función que usamos en caso de que si haya un recorrido posible del nodo inicial al final para ir recorriendo del nodo final a su anterior, y de este a su anterior hasta llegar al nodo inicial. Por cada nodo que se va pasando se va guardando en un array, que es el que finalmente devuelve la función.
- Y por último, la función `estaCerrada(fila, col)`, que lo que hace es, dada una fila y una columna comprueba que para todas las coordenadas de array de nodos cerrados si alguno tiene esa posición, entonces significa que el nodo que esté en esa fila y en esa columna ya ha sido cerrada. En caso contrario devuelve false.

La representación de los distintos elementos del tablero son: las casillas vacías son celdas azules, la casilla de inicio es verde oscuro, la casilla de fin es roja, los obstáculos son negros, y los way points son verdes claros con un número en el centro que los distingue por orden de inclusión en el tablero. Este diseño se puede apreciar en el apartado de ejemplos de uso.

AMPLIACIONES

Una de las ampliaciones que decidimos hacer son los llamados “way points”. No requiere demasiada modificación del código. Se creó una cola en la cual se añadían en orden los “way point”. Después solo hay que resolver el algoritmo A* para cada uno de ellos cambiando el inicio y final por estos “way points”. Por último se genera un array de estas soluciones independientes y se imprime.

Se añade un bucle a la función *algoritmo()* el cual repite el algoritmo hasta que no queden “way points”. Aparte se creó una nueva función llamada *loopWayPoint()* la cual *resetea la información del tablero aparte de intercambiar ini por el fin actual y sacar a continuación el siguiente fin de la cola de “way points”*.

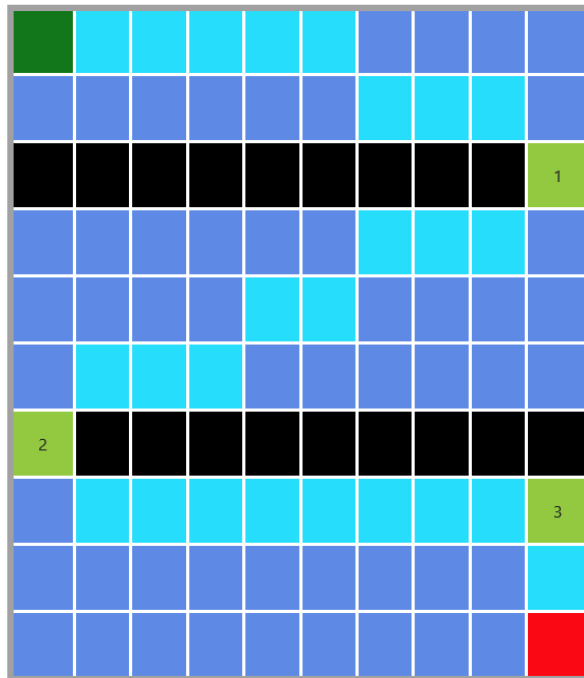


imagen 2

Como se puede apreciar en la imagen 2 el algoritmo pasa en el orden de los way point están marcados en ese orden en concreto.

MANUAL DE USUARIO

En la carpeta subida a la entrega de la práctica consta de una carpeta con todos los ficheros necesarios para la ejecución del algoritmo (posteriormente explicamos cómo ejecutarlo) junto con la memoria de la práctica en formato pdf.

Para la ejecución del algoritmo y su uso no hace falta ninguna descarga previa o inclusión de una librería externa, ya que los únicos recursos externos que hemos utilizado han sido la librería de bootstrap y la de jquery, y ambos están importados a través de links en el archivo index.html, por tanto necesitamos tener una conexión a internet para el correcto funcionamiento del algoritmo. Para la ejecución únicamente es necesario un navegador web en el que abrir el archivo index.html (ver imagen 3.1), que está dentro de la carpeta de la entrega.








	.vscode	11/03/2022 14:17	Carpeta de archivos	
	AEstrella	11/03/2022 14:48	Archivo JavaScript	5 KB
	app	11/03/2022 19:48	Archivo JavaScript	5 KB
	Cordenada	11/03/2022 13:07	Archivo JavaScript	2 KB
	index	11/03/2022 12:49	Microsoft Edge HT...	2 KB
	PriorityQueue	11/03/2022 13:56	Archivo JavaScript	2 KB
	style	11/03/2022 12:49	Documento de hoj...	3 KB

imagen 3.1

Abrir dicho archivo con cualquier navegador que pueda ejecutar javascript (nosotros hemos usado google chrome) o haciendo doble click en el archivo, en cuyo caso el fichero se abrirá con el navegador por defecto del ordenador. Una vez abierto en el navegador el algoritmo está listo para ser ejecutado.

Primero habrá que marcar obligatoriamente una casilla de inicio y otra de fin (o al revés) seleccionando en el panel de control una de las distintas opciones, posteriormente se marcará con el ratón en la casilla que se quiera situar el inicio o el final. Y de la misma forma opcionalmente se podrán marcar obstáculos o way points (ver imagen 3.2). Una vez situados el inicio y el fin, para ejecutar el algoritmo se pulsará en el botón azul “calcular” y se mostrará el camino solución en el tablero (ver imagen 3.3).

☐ Seleccionar ubicación de origen
☒ Seleccionar ubicación de final
☐ Seleccionar ubicación obstaculos
☐ Seleccionar ubicación "way point"

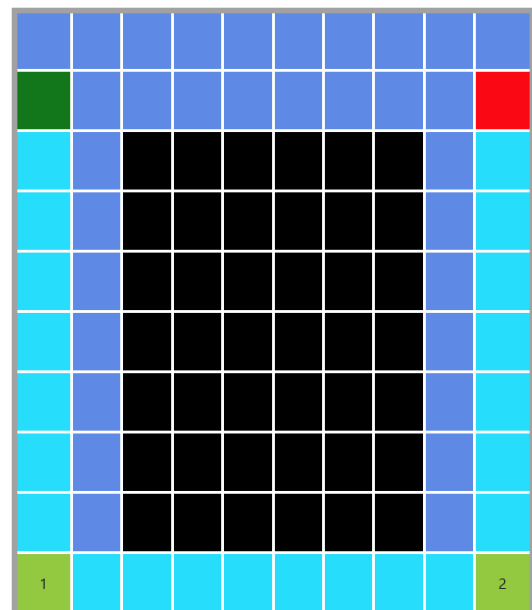
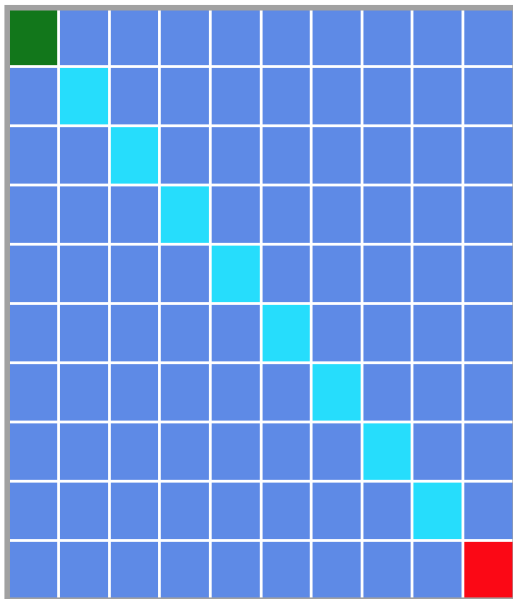
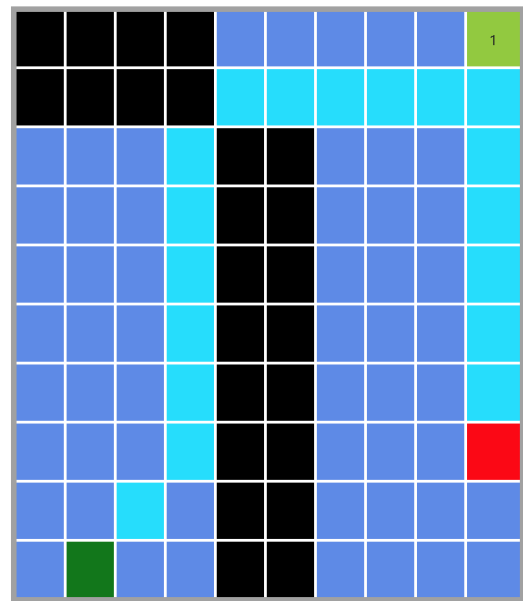
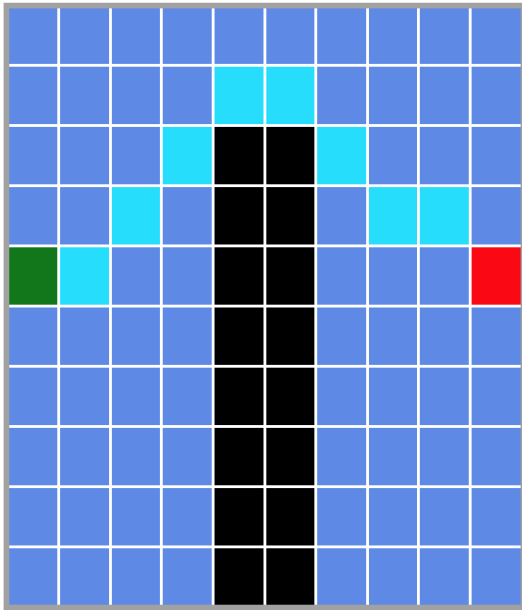
imagen 3.2

Una vez finalizado el algoritmo si se quiere volver a repetirlo con distinta ordenación del inicio, del final y de los obstáculos basta con pulsar en el botón rojo “resetear” y el tablero volverá a su estado inicial (ver imagen 3.3).



imagen 3.3

EJEMPLOS DE USO



Aquí ponemos unos pequeños ejemplos de uso para ver el funcionamiento del algoritmo. Para el caso en el que no haya solución posible debido a que el nodo final sea inaccesible, no se imprimirá nada en el tablero.